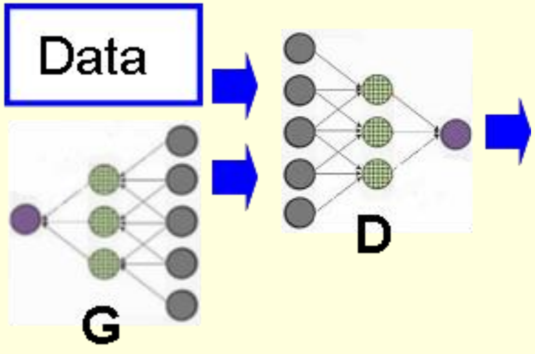
# Deep Learning by Example on Biowulf

## Class #4. Generative Adversarial Networks (GANs) and their application to biological data synthesis

Gennady Denisov, PhD

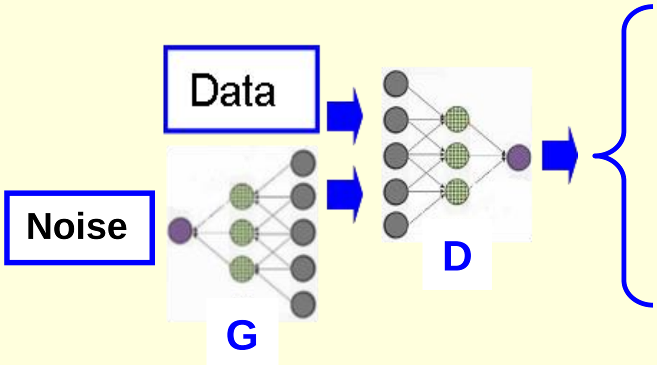


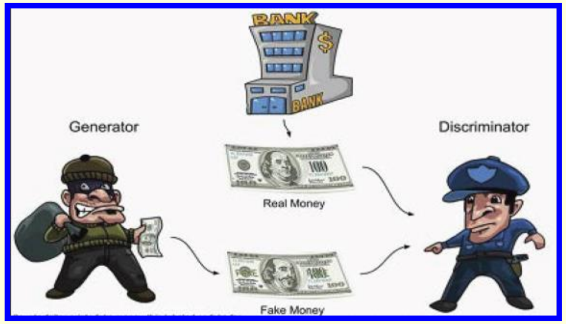
# Intro and goals

*I.Goodfellow et al., Generative Adversarial Nets. NIPS Proc. 2014*

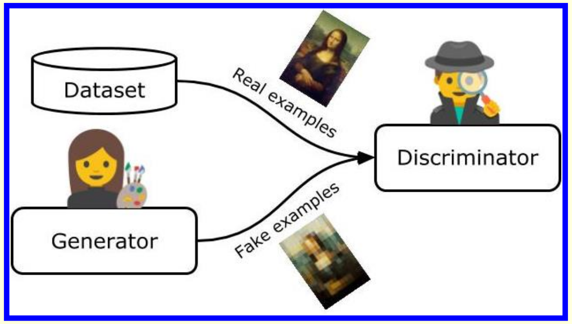
## What is a GAN?

- A composite network comprising 2 subnetworks: **G**enerator and **D**iscriminator
- The **G** produces fake data from scratch/noise; learns to **trick** the **D**
- The **D** compares fake data against the true data; learns to **expose** the **G**




**Counterfeiter vs police analogy**




**Art forger vs critique analogy**

## Features:

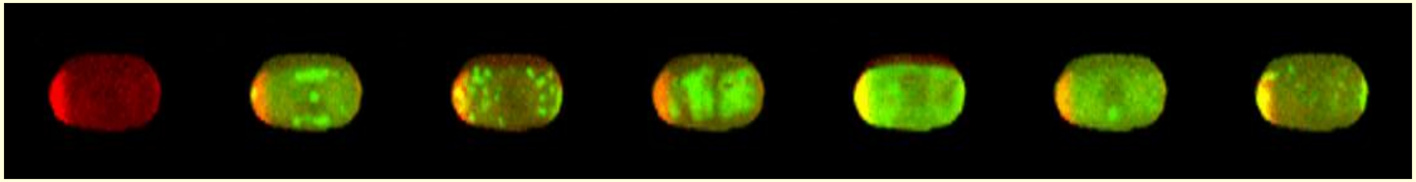
**Generative model:** the goal is to generate new, **synthetic** instances of data that can pass for real data

**G** and **D** are trained by **pitting** one against the other – thus the **adversarial**, i.e. antagonistic, or confrontational

## Examples:

Generating face images

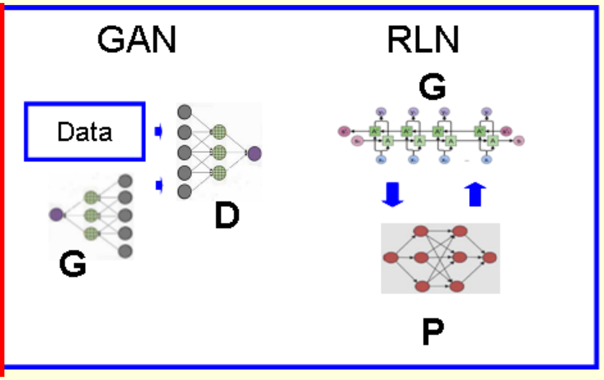
**BioGANs: GANs for biological image synthesis**



# Examples overview

| # | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Biological Appliation | Bioimage segmentation/ fly brain connectome project | Genomics/ predicting the function of non-coding DNA (~98%) | Genomics/ clustering of cancer samples based on gene expression (~2%) | Bioimage synthesis / developmental biology | Small drug molecule design |
| Network type | Convolutional Neural Network | Recurrent Neural Network | Auto-encoder | Generative Adversarial Network | Reinforcement Learning Network |
| ML type | Supervised | Supervised | Unsupervised | Unsupervised | Reinforcement |

- unsupervised ML
- generative model, functionally similar to VAE
- composite network comprising two subnetworks
- the two subnetworks are trained interactively, by playing a minimax game
- GAN flavors: GAN, DCGAN, WGAN, WGAN-GP,…

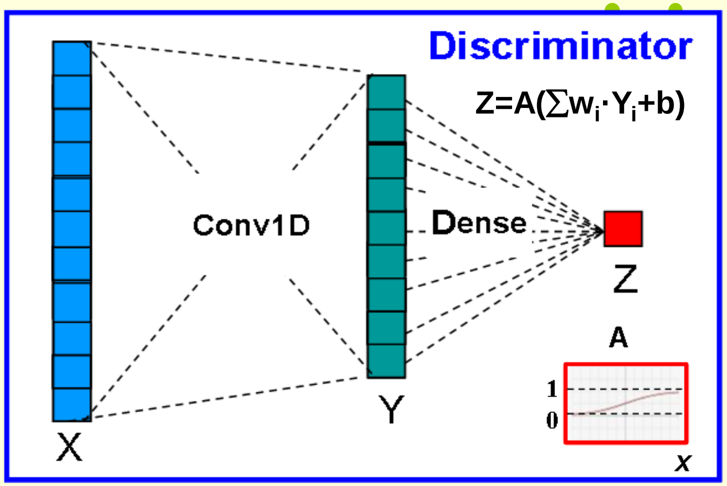


# Deep Convolutional GAN (DCGAN): a simple example

**tensors, units, layers, parameters, hyperparameters, convolution**

*A.Radford et al, arXiv:1511.06434v2 (2016)*

## RNN/1D CNN prototype example from class #2:

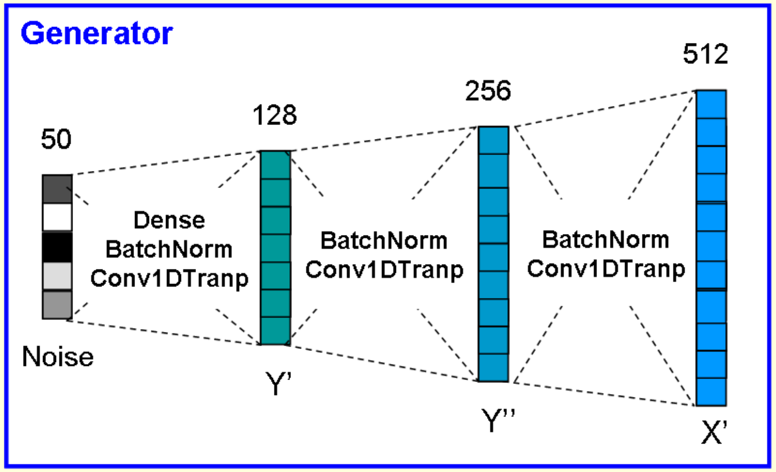



**Input:** a set of training sequences of 0's and 1's with **binary labels** assigned depending on whether or not a certain (unknown) **motif** is present

**Example**: 01011100101

**Task**: predict the label, or the occurrence of the **unknown** motif, in new, previously unseen sequences.

## DCGAN prototype example:




**Input:** a training set of only "good" sequences of 0's and 1's, i.e. **all** of them contain a certain motif

**Example**: 0101100110011001100111

**Task**: learn what makes all of the training sequences "good" and then generate new "good" sequences from scratch.

**Challenge:** no labels; only positive examples.

## Architecture guidelines for stable DCGANs:

- Use **convolutions** (D) and **transposed convolutions** (G) instead of pooling layers
- Use **BatchNormalization** in both the G and the D.
- Avoid Dense/Fully Connected hidden layers
- ReLU activation in G for all layers except for the output and LeakyReLU activation in D.

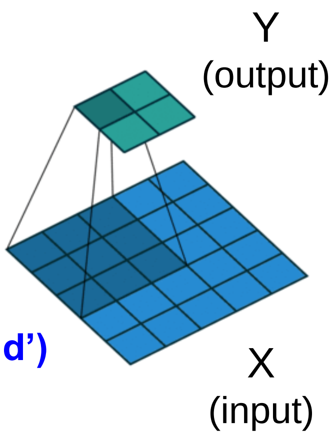
# The transposed convolution (a.k.a. deconvolution, or fractional-strided convolution)

**convolution, transposed convolution, stride, kernel size, padding**

*V.Dumoulin, F.Visin - A guide to convolution arithmetic for deep learning (2018)*

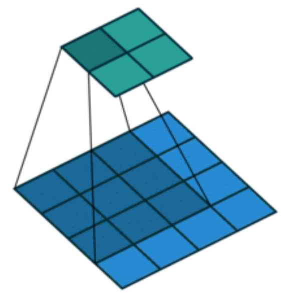
**Conv2D**

**input size i = 5**
**output size o = 2**
**kernel_size k = 3**
**strides s = 2**
**padding p = 0 ('valid')**



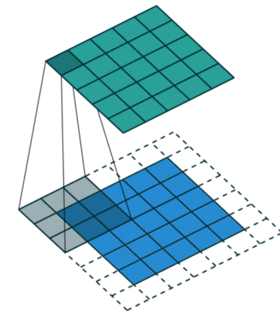

**Conv2D**

**i = 4**
**o = 2**
**k = 3**
**s = 1**
**p = 0**



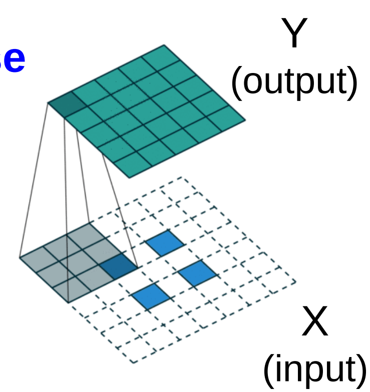
**Conv2D**

**i = 5**
**o = 2**
**k = 3**
**s = 1**
**p = 1('same')**



**Conv2DTranspose**

**input_size i' = 2**
**output_size o' = 5**
**kernel_size k' = 3**
**strides s' = 2**
**padding p' = 2**




**Conv2D:** $\boxed{i + 2*p = k + s*(o - 1)}$

**'valid' padding: p = 0**
**'same' padding: o = round(i / s)**

**Conv2DTranspose:** $\boxed{o' = i' + (i'-1)*(s'-1) + 2*p' - k' + 1)}$

**'valid' padding: p' = k' – 1**
**'same' padding: o' = i' * s'**

# The simple GAN training code: (1) header, (2) getting data and (3) defining a model

### motif, discriminator, compile, loss, optimizer

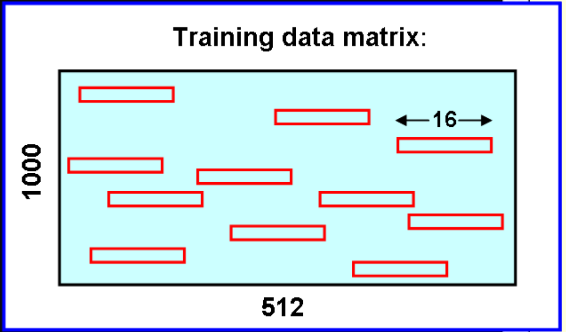
**(1) Header:**
- general Python imports
- Numpy imports
- Keras library imports

**(2) Get data**
- motif
- noise_len

**(3) Define a model**
- discriminator (D)

denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro

```
#!/usr/bin/env python

import re, random, string
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Conv1D, Flatten, Reshape, Lambda, \
                         Conv2DTranspose, BatchNormalization, Activation
from keras.optimizers import RMSprop
import keras.backend as K

# Get training data
motif           = "1100110011001100"
num_train_data  = 1000
seq_len         = 512
noise_len       = 50
epochs          = 3000
b_size          = 100      # batch_size
n_chan          = 10       # number of channels
f_size          = 3        # filter size

np.random.seed(1)
x_str = [''.join([random.choice('01') for i in range(seq_len)]) \
                                      for j in range(num_train_data)]
for j in range(num_train_data):
    rint = np.random.randint(0, high=seq_len-len(motif))
    x_str[j]= x_str[j][:rint] + motif + x_str[j][(rint+len(motif)):]
x_train = np.reshape(np.array([[int(c) for c in x_str[j]] \
          for j in range(num_train_data)]), [num_train_data,seq_len,1])

# Define a model
D = Sequential()
D.add(Conv1D(n_chan, f_size, activation='relu', input_shape=(seq_len,1)))
D.add(Flatten())
D.add(Dense(1, activation='sigmoid'))
D.compile(loss='binary_crossentropy', optimizer= RMSprop(lr=0.0001))
                                                                   35,73          Top
```



# The simple GAN training code: (3) defining a model (cont.) and (4) running the model

**Lambda, BatchNormalization, Conv1DTranspose, generator, trainable, combined_model, train_on_batch, epoch, save_weights**

**Lambda layer:**
Wrap an arbitrary mathematical expression into Keras layer

**BatchNormalization layer**

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

**(3) Define a model**
- Conv1DTranspose
- Lambda
- BatchNormalization
- generator (G)
- combined model (GAN)

GAN = D(G(z))

**(4) Run the model**
- train_on_batch,
- epoch
- save_weights

denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro

```
# Define a model
def Conv1DTranspose(inp, nf, ks, s=2, p='same'):
    x1 = Lambda(lambda x: K.expand_dims(x, axis=2))(inp)
    x2 = Conv2DTranspose(filters=nf,kernel_size=(ks,1),strides=(s,1),padding=p)(x1)
    return Lambda(lambda x: K.squeeze(x, axis=2))(x2)

G = Sequential()
G.add(Dense(int(seq_len/8)*n_chan, input_shape=(noise_len,)))
G.add(Reshape((int(seq_len/8), n_chan)))
G.add(BatchNormalization(momentum=0.8, epsilon=1.e-5))
for i in range(0, 2):
    G.add(Lambda(lambda x: Conv1DTranspose(x,n_chan,f_size)))
    G.add(BatchNormalization(momentum=0.8, epsilon=1.e-5))
G.add(Lambda(lambda x: Conv1DTranspose(x,1,3)))
G.add(Activation('sigmoid'))
G.summary()

GAN = Sequential()
GAN.add(G)
D.trainable = False
GAN.add(D)
GAN.compile(loss='binary_crossentropy', optimizer=RMSprop(lr=0.001))

# Run the model on the data
for epoch in range(epochs+1):
    true_data = np.array(x_train[np.random.randint(0, x_train.shape[0], b_size)])
    true_data = np.reshape(true_data, (true_data.shape[0], true_data.shape[1], 1))
    noise     =          np.random.normal(0, 1, (b_size, noise_len))
    fake_data = G.predict(noise)
    D_loss_true = D.train_on_batch(true_data, np.ones( (b_size,1)))
    D_loss_fake = D.train_on_batch(fake_data, np.zeros((b_size,1)))
    G_loss      = GAN.train_on_batch(noise, np.ones( (b_size,1)))
    if epoch%100 == 0:
        print ("%d: D_loss %20.16f G_loss %20.16f" % \
              (epoch,(D_loss_true + D_loss_fake)/2.,G_loss))
G.save_weights("weights.generator.h5")
                                                        36,58          All
```

**train_on_batch, one epoch:**
- train D: D(true data) → ones
- train D: D(fake data) → zeros (= train GAN with G weights frozen)
- train G: G(fake_data) → ones (= train GAN with D weights frozen)

# The GAN optimization objective

*X.Chen et al, arXiv:1802.01765v1 (2018)*

D(Real) $\to$ 1
D(Fake) $\to$ 0

**Discriminator:** $\log D(\underbrace{x}_{\text{Real data}}) + \log(1 - D(\underbrace{G(z)}_{\text{Fake data}})) \to$ **max**

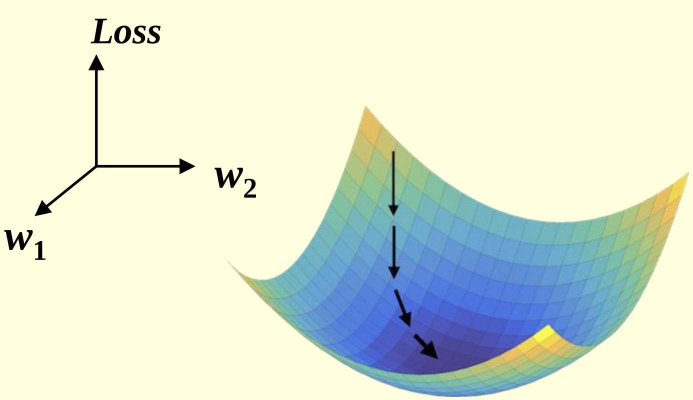
G(Fake) $\to$ 1

**Generator:** $\log(1 - D(G(z))) \to$ **min**

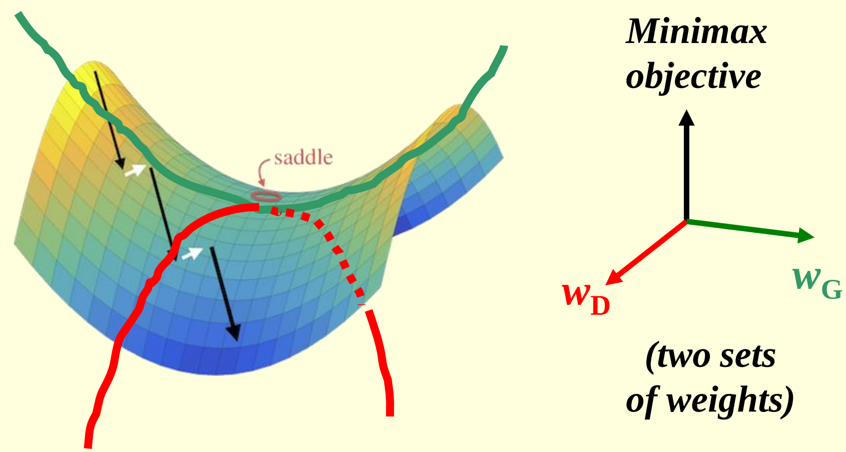
**The minimax optimization objective:**

$$\boxed{\min_G \max_D E_{data}\{\log D(x)\} + E_{noise}\{\log(1 - D(G(z)))\}}$$

**x = data**
**z = noise**




Standard neural net




Adversarial neural net

# The simple GAN prediction code

## load_weights, predict

**Header:**
- general Python imports
- Numpy imports

…

**Get data**

**Define a model**

**Run the model**
- load_weights
- noise
- predict

Select denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro

```
#!/usr/bin/env python

import re, random, string
import numpy as np
...

# Get testing data
motif           = "1100110011001100"
num_test_data   = 1000
...
np.random.seed(1)
x_test = [''.join([random.choice('01')  for i in range(seq_len)]) \
                                        for j in range(num_test_data)]
# Define a model
...

# Run the model
G.load_weights("weights.gan_generator.h5")
G_count = 0
R_count = 0
re.compile(motif)
for i in range(num_test_data):
    noise = np.random.normal(0, 1, (1, noise_len))
    generated_seq = "".join([str(int(s)) for s in \
                            np.round(np.squeeze(G.predict(noise)))])
    G_pos = re.search(motif, generated_seq)
    if G_pos:
        G_count += 1
    random_seq = x_test[i]
#   print("dir(G_pos)=", dir(G_pos))
    if re.search(motif, random_seq):
        R_count += 1
    print(str(i) + ": G_count=", G_count, "motif_start=", \
          G_pos.start(), " R_count=", R_count)

print("Motif count in generated sequence: " + str(G_count) + \
          "/" + str(num_test_data))
print("Motif count in random    sequence: " + str(R_count) + \
          "/" + str(num_test_data))
                                                          39,46          All
```

# How to run the simple GAN application on Biowulf?

denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro

```
$ sinteractive --gres=gpu:p100:1 --mem=4g

$ module load DLBio/class4
...

$ ls $DLBIO_BIN
simple_gan_train.py  simple_gan_predict.py   simple_wgan_train.py

$ simple_gan_train.py
Using TensorFlow backend.
...
100: D_loss   0.5485115051269531 G_loss   0.2828627526760101
200: D_loss   0.3392027914524078 G_loss   0.0490138642489910
300: D_loss   0.1760229766368866 G_loss   0.0042200786992908
...
3000: D_loss   0.0000000026483797 G_loss   0.0000000000000000

$ ls weights.gan_generator.h5
weights.gan_ator.h5

$ simple_gan_predict.py
...
0: G_count= 1 motif_start= 21  R_count= 0
...
726: G_count= 727 motif_start= 7  R_count= 9
727: G_count= 728 motif_start= 111  R_count= 9
...
999: G_count= 1000 motif_start= 39  R_count= 10
Motif count in generated sequence: 1000/1000
Motif count in random    sequence: 10/1000

$ simple_wgan_train.py
...
100: C_loss  -0.0327724255621433 G_loss  -0.0370155312120914
200: C_loss  -0.0348766297101974 G_loss  -0.0597182549536228
...
                                                  36,55          Top
```

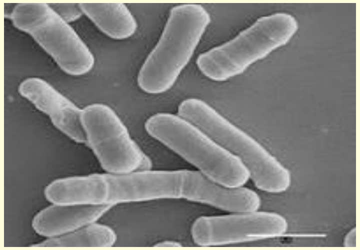
**Executables** — ls $DLBIO_BIN

**A checkpoint file** — weights.gan_generator.h5

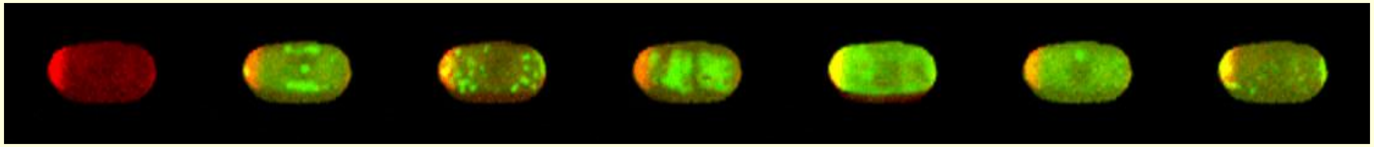
**Counts of motifs produced by Generator or randomly** — Motif count lines

# Example 4. BioGANs: GANs for Biological Image Synthesis

*A.Osokin e.a. IEEE Int. Conf. on Computer Vision (ICCV), 2017*
*https://github.com/aosokin/biogans*
*https://hpc.nih.gov/apps/biogans.html*



Fission yeast cells



Bgs4 | Alp14 + Bgs4 | Arp3 + Bgs4 | Cki2 + Bgs4 | Mkh1 + Bgs4 | Sid2 + Bgs4 | Twa1 + Bgs4

**Biological task:** investigate how the **polarity factors** **interact** with one another

**Computational task:** train a GAN on available data and generate **synthetic** images that visualize a **synchronized** distribution of **multiple** **polarity factors**, together with **growth factor Bgs4** at the **same stage** of a cell cycle (i.e. the data that cannot be produced experimentally)

**Data:** the Localization Interdependency network (**LIN**) dataset

**The BioGANs pipeline (reimplemented in Keras from PyTorch):**

**biogans_train.py** → **biogans_predict.py** → **biogans_visualize.py**

# An overview of the BioGANs training code

*https://hpc.nih.gov/apps/biogans.html*

**The Keras source code:**
**biogans_train.py**
**biogans.predict.py**
**biogans_visualize.py**
options.py, dataloader.py,
models.py, gans.py

**Header**
- import statements
- parsing the command line options

**Getting data**
- LIN dataset

**Define a (network) model**
- models available:
  DCGAN,
  DCGAN-separable,
  DCGAN-starshaped

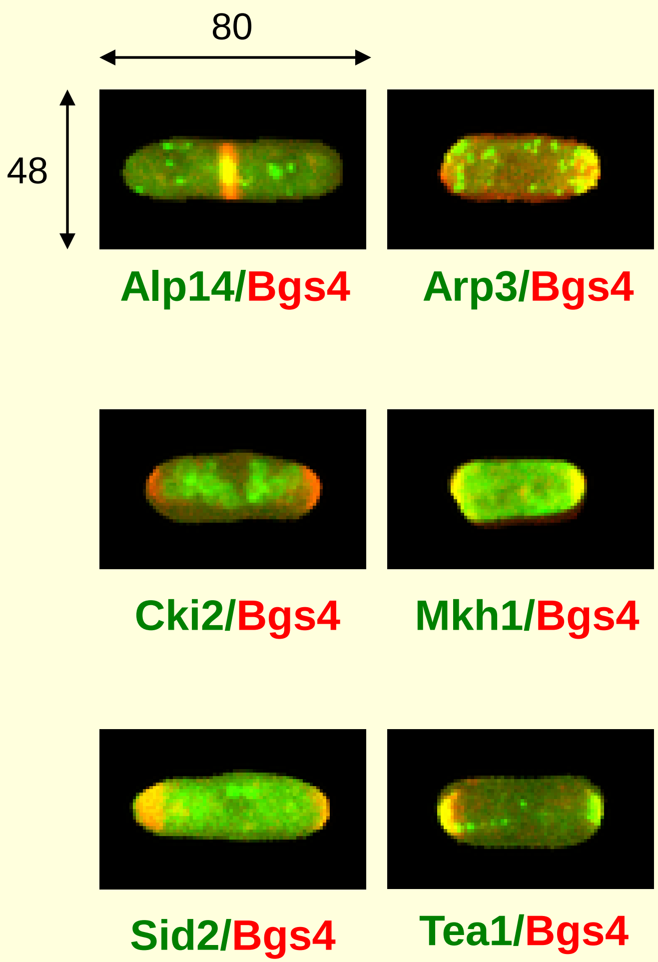
**Run the model**
- GAN algorithms:
  (traditional) GAN
  WGAN
  WGAN-GP
- optimizer: RMSProp

denisovga@biowulf:/data/denisovga/1_DL_Course/4_GANs

```
#!/usr/bin/env python

import os, sys, random
import numpy as np
import gans
from dataloader import get_data
from options import parse_training_arguments, process_options
from models import get_network_models

import tensorflow as tf
from tensorflow.keras.optimizers import Adam, RMSprop

# ----------------------------------------------------------------

if __name__ == '__main__':
    opt = parse_training_arguments()
    opt, DCGAN_model, gan_algorithm, optimizer = process_options("train", opt)

    # Load data
    dataset, opt.n_classes = get_data(opt, "train")

    # Define a model
    os.environ['CUDA_VISIBLE_DEVICES'] = "0"
    if opt.num_gpus > 1:
        for j in range(1, opt.num_gpus):
            os.environ['CUDA_VISIBLE_DEVICES'] += "," + str(j)
    with tf.device('/cpu:0'):
        random.seed(opt.random_seed) # fix random seed
        netG, netD = get_network_models(DCGAN_model, opt, opt.red_portion)

    # Run the model
    if gan_algorithm == "GAN":
        gans.GAN(    netG, netD, opt).train(dataset, opt)
    elif gan_algorithm == "WGAN":
        gans.WGAN(    netG, netD, opt).train(dataset, opt)
    elif gan_algorithm == "WGAN-GP":
        gans.WGAN_GP(    netG, netD, opt).train(dataset, opt)
    else:
        sys.exit("Undefined gan_algorithm: " + gan_algorithm + "\n")
                                                          39,9          All
```

# BioGANs data: the Localization Interdependency Network (LIN) dataset

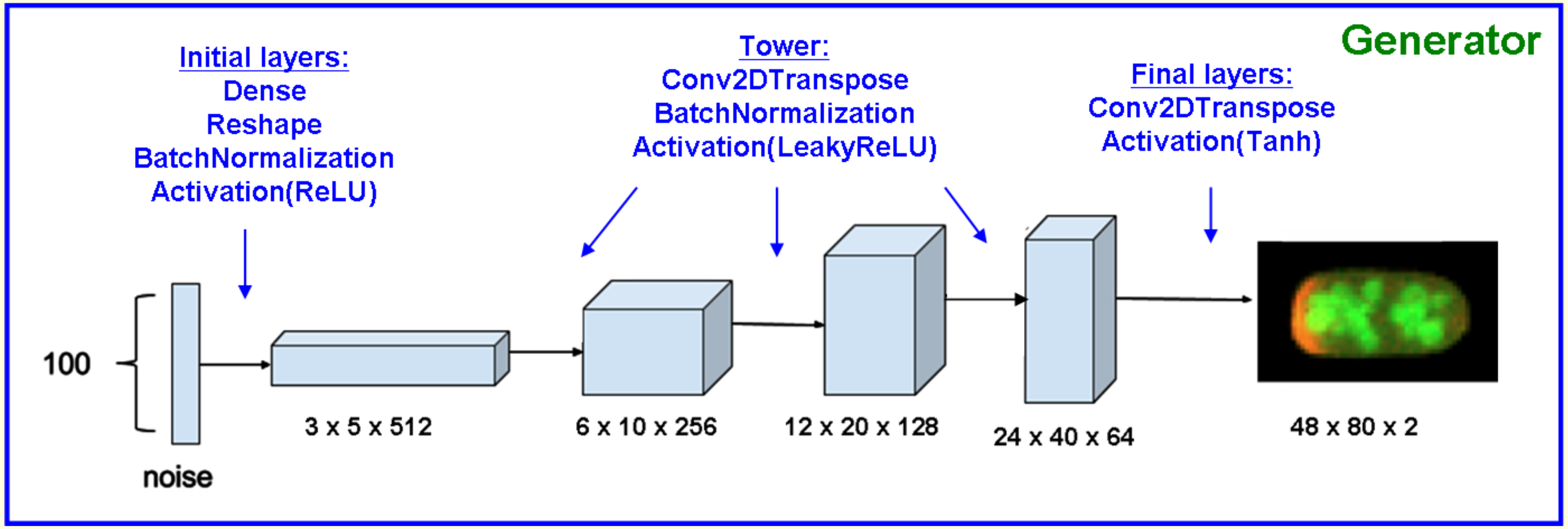
*J.Dodgson et al, https://www.biorxiv.org/content/10.1101/116749v1.full*

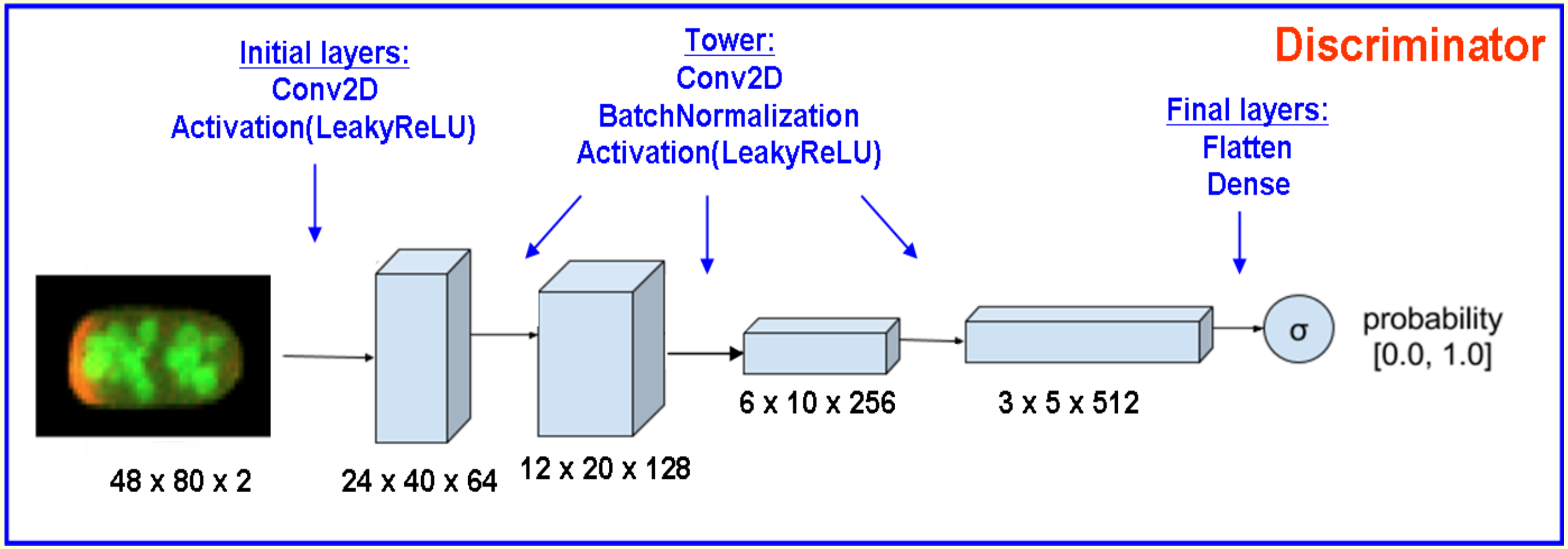



## Features:

- **2D fluorescence microscopy images** of Fission yeast cells, each (7 ÷14) x 4 μm
- 2-channel images of size is **48 x 80 pixels** (1 pixel = 100 nm)
- **red** channel = protein **Bgs4,** localizes in the **area of active growth**
- **green** channel = any of **41** different **polarity factors** that **define a cell geometry**
- **170,000 images** for **41** polarity factors available in the in the LIN dataset.
- the BioGANs application focuses on **Bgs4** and **6 polarity factors Alp14, Arp3, Cki2, Mkh1, Sid2** and **Tea1,** with total **26,909 images**

# The BioGANs DCGAN (i.e. basic) model



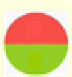



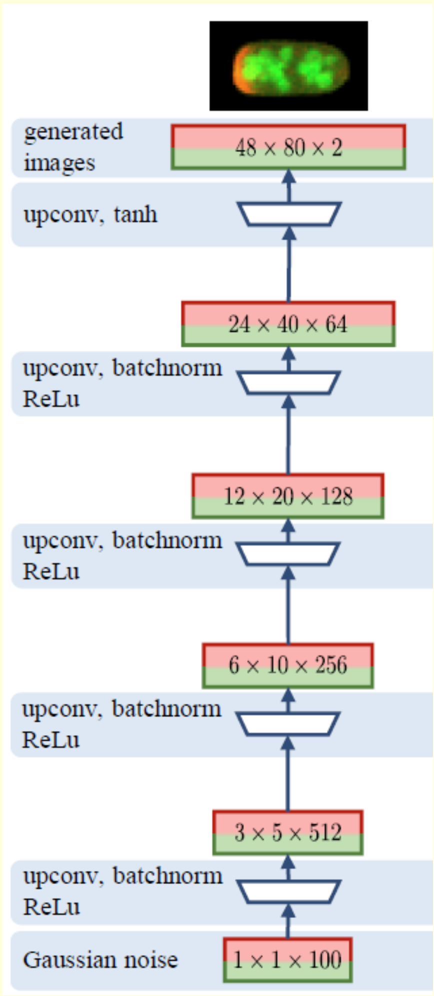
# BioGANs generator architrectures: DCGAN, DCGAN-separable and DCGAN-starshaped

*A.Osokin e.a. IEEE Int. Conf. on Computer Vision (ICCV), 2017*

**How to generate multiple green channels given a signal in a red channel?**

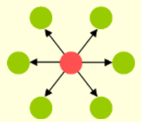
**DCGAN**

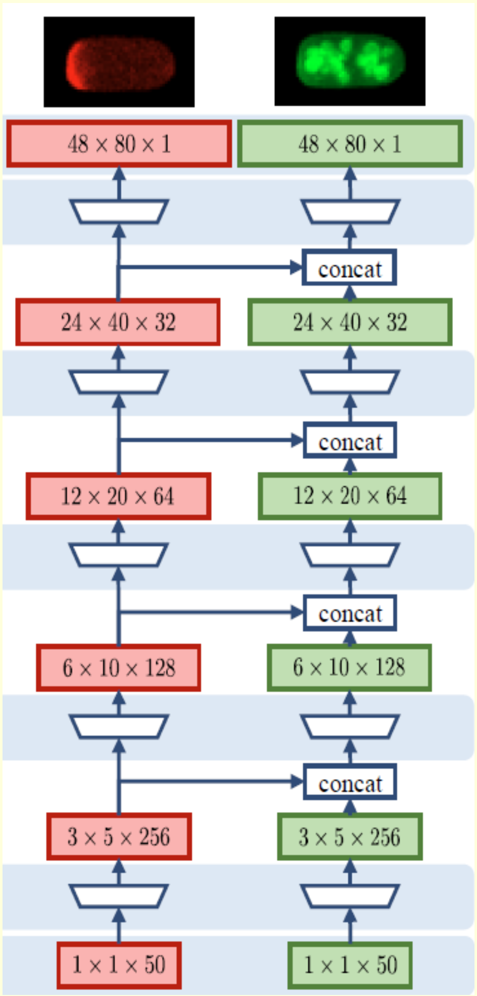





**DCGAN-separable**

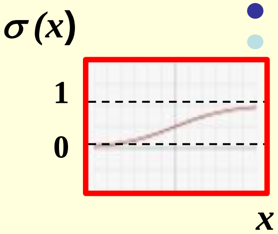
**DCGAN-starshaped**





# The Wasserstein GAN (WGAN)

*M.Arjovsky et al, Wasserstein GAN – arXiv: 1701.07875 (2017)*



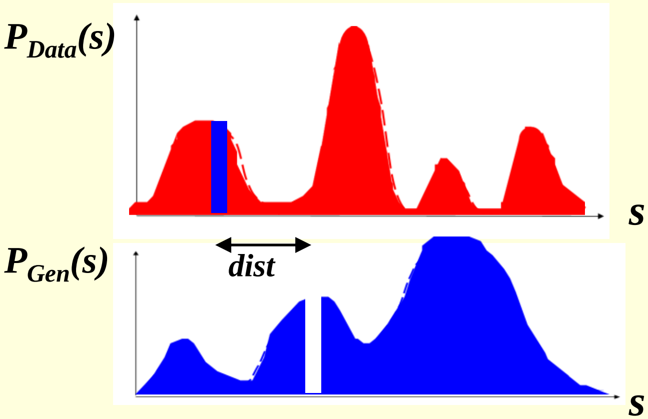

**Problem with traditional GAN: vanishing gradients due to the last/sigmoid layer in the Discriminator:**

$D(I, w) = \sigma(F(I, w)) \;\; \Rightarrow \;\; \nabla_w D = \sigma' \cdot \nabla_w F \rightarrow 0$ **at saturation**

**WGAN ideas:**

- **get rid of the $\sigma$ layer => can no longer use the BCE loss; the $D$ becomes $F$**
- **rename $F$ to critic: it will output a score $s$, not a probability**
- **use the Earth Mover's distance (EMD) between the distributions of the critic scores $P_{Data}(s)$ and $P_{Gen}(s)$ as a new loss function**

**EMD, a.k.a. Wasserstein loss = <u>minimum</u> amount of work to transform one distribution to another**




**Work =** $dist * mass$ *(=area of the piece)*

**Binary cross entropy loss:**

$$BCE = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p_i(w)) + (1-y_i) \cdot log(1 - p_i(w))$$

**Wasserstein loss:**

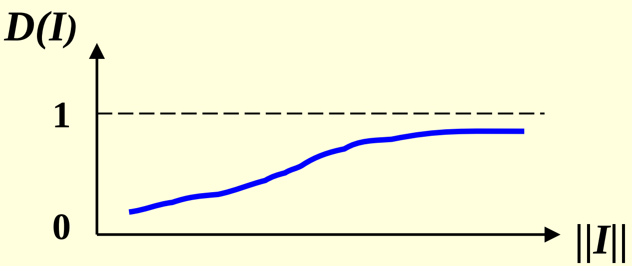
$$EMD \approx -E\,[s^{Data} \cdot s^{Gen}]$$

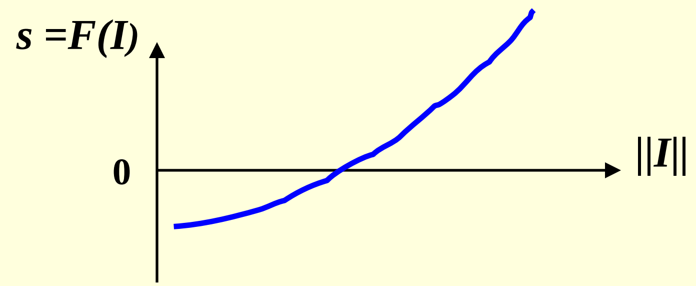
$EMD \rightarrow$ min forces the two distributions to have maxima at the same locations

# WGAN vs WGAN with gradient penalty (WGAN-GP)

*Gulrajani et al., Improved Training of Wasserstein GANs - arXiv:1704.00028v3 (2017)*

**How can we limit the growth of *F* to avoid instability?**




**(Traditional) GAN:**
***use sigmoid activation:*** $D(I) = \sigma(F(I))$

**WGAN:**
***clip the weights that are beyond*** $[-c, c]$

Data transformation by one layer:

$$Z = A(\sum w_i \cdot X_i + b)$$

**WGAN-GP:**
***penalize the gradient (i.e. slope of F)***

**WGAN features:**
(1) use EMD loss
(2) **clip <u>all weights</u> after <u>each epoch</u>**
**(usually, $c$ = 0.01)**
(3) rename Discriminator to Critic
(4) use RMSProp optimizer with lr = 0.00005

**WGAN-GP features:**
(1), (3), (4)
**(2) penalize the gradient (usually, $\lambda$ = 10)**

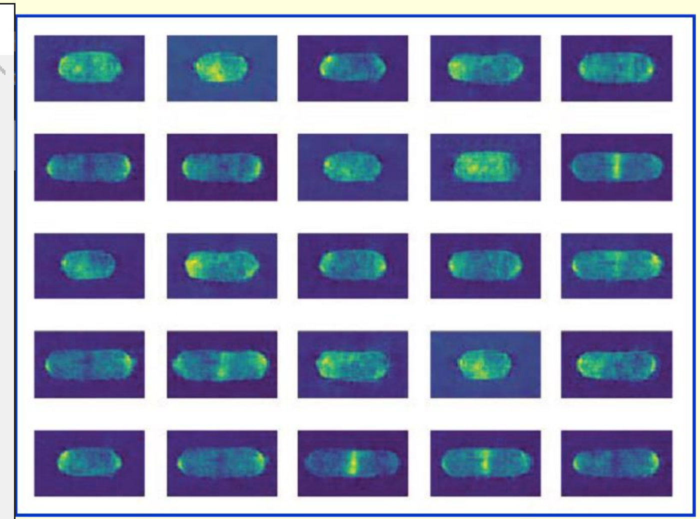
$$\textit{WGAN-GP loss} = EMD + \lambda \cdot \|\nabla F\|$$

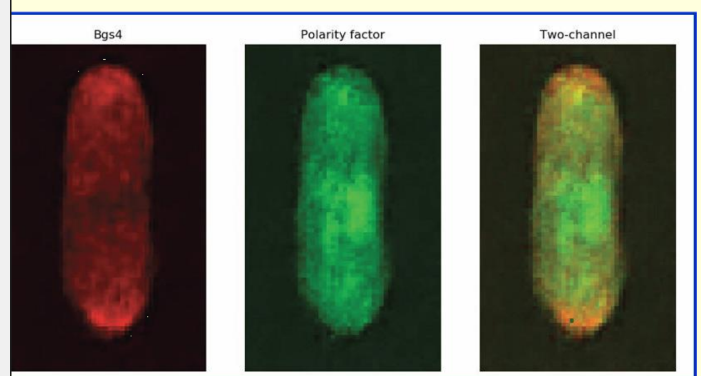
# How to run the BioGANs application on Biowulf?

*https://hpc.nih.gov/apps/biogans.html*

denisovga@biowulf:/data/denisovga/1_DL_Course/4_GANs

```
sinteractive --mem=40g --gres=gpu:p100:1,lscratch:10

module load biogans

cp $BIOGANS_DATA/* .

ls $BIOGANS_SRC
biogans_predict.py  biogans_visualize.py  gans.py    options.py   utils.py
biogans_train.py    dataloader.py         models.py  __pycache__

biogans_train.py -d <data_folder> [-m <network_model>] [-a <gan_algorithm>]

# NOTES:
# network_model = DCGAN (default), DCGAN-separable or DCGAN-starshaped
# gan_algorithm =   GAN (default), WGAN or WGAN-GP

Example:
biogans_train.py -d data/LIN_Normalized_WT_size-48-80_train/Alp14

biogans_predict.py -i <input_file> [ other options ]

Example:
biogans_predict.py -i checkpoints/model.generator.Alp14.DCGAN.GAN.1.h5

biogans_visualize.py -i <input_file>

Example:
biogans_visualize.py -i checkpoints/model.generator.Alp14.DCGAN.GAN.1.h5

                                                            29,44         All
```





# The Root Mean Squared propagation (RMSprop) optimizer

*Slides: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf*
*Video: https://goo.gl/XUbIyJ*

$$\boxed{w_{t+1} = w_t - \gamma \cdot \nabla_w J(w_t)}$$

Basic gradient descent formula for updating weights

$w$ = vector of weights
$t$ = update #
$\gamma$ = learning rate ( a hyperparameter)
$\nabla_w J$ = gradient of the loss with respect to weights

effective $\gamma$

$$w_{t+1} = w_t - \frac{\gamma}{\sqrt{E[\nabla_w J_w(w_t)^2] + \varepsilon}} \cdot \nabla_w J(w_t)$$

The RMSprop-based formula for updating weights

$E[\ldots]$ = running average of the magnitudes of recent gradient squares
$\varepsilon$ = small parameter

How to compute the running average:

$$\boxed{E[\nabla_w Jw(w)^2]_t = \beta \cdot E[\nabla_w Jw(w)^2]_{t-1} + (1 - \beta) \cdot \nabla_w J(w_t)^2}$$

$\beta \sim 0.9$

# Conclusions

1) Intro using a simple example
- simple GAN that synthesizes a sequence containing a certain motif:
  Discriminator is the same as the network from class #2
  Generator network produces a sequence from random noise
- the Conv2DTranspose (transposed convolution, a.k.a. deconvolution) layer
- the BatchNormalization layer
- the train_on_batch method

2) The BioGANs application:
- DCGAN, DCGAN-separable and DCGAN-starshaped models
- WGAN (Wasserstein GAN ) and the Earth Mover's distance (EMD)
- WGAN-GP: the Wasserstein GAN with gradient penalty loss

3) Other topics:
- the gradient descent-based optimization algorithm RMSprop